RTO-RTDB: A REAL-TIME OBJECT-ORIENTED DATABASE MODEL

Nada Louati and Rafik Bouaziz
MIRACL-ISIMS, Sfax University, BP 1088
3018 Sfax, Tunisia
{nada.louati, raf.bouaziz}@fsegs.rnu.tn

Claude Duvallet and Bruno Sadeg
LITIS, UFR des Sciences et Techniques, BP 540
76 058 Le Havre Cedex, France
{claude.duvallet, bruno.sadeg}@univ-lehavre.fr

ABSTRACT
In recent years search for proper extensions of the object model to suit real-time database community has become a critical research issue. In this paper, we present such an extension of the object model, called RTO-RTDB. A concrete static and dynamic views of the model are presented. A major attraction of this model is in that it is effective not only in the use of the feedback control real-time scheduling theory but also in the accurate representation of real-time databases properties. It is used to demonstrate how to separate structural and concurrency concerns ensuring also high-level abstraction for parallelism modeling. In order to improve and to facilitate the RTO-RTDB object model specification, we have used the MARTE and UML-RTDB profiles as UML extensions to describe real-time aspects on one hand, and to express real-time database features on the other hand.

KEY WORDS
real-time, database, object-model, feedback control

1 Introduction

Real-**T**ime **D**ata**B**ases (RTDBs) are typically used to manage environmental data in computer control applications, such as air traffic control, automated manufacturing, and military command and control [1]. RTDBs manipulate real-time data and execute real-time transactions [1]. Real-time data are divided into two types: sensor data and derived data. Sensor data are issued from sensors, whereas derived data, they are computed from sensor data. Real-time transactions are classified into two categories: update transactions and user transactions [2]. Update transactions are used to update values of real-time data in order to reflect the state of the real world. They are executed periodically to update sensor data, or sporadically to update derived data. User transactions arrive aperiodically.

RTDBs have all requirements of traditional databases, such as the management of accesses to structured, shared and permanent data, and they require management of time-constrained data and time-constrained transactions [3]. The methods available to describe the conceptual data model of traditional database can not be directly applied to describe the conceptual data model of a RTDB, since there is no mechanism to deal with the representation of time constraints. Moreover, generally speaking, RTDBs are very complex, leading to an extensive conceptual description, and to a prohibitive complexity from the practical point of view. In this paper, we address this problem by defining a real-time object-oriented database model called RTO-RTDB (**R**eal-**T**ime **O**bject for **R**eal-**T**ime **D**ata**B**ases), which encapsulate time-constrained data, time-constrained transactions, and concurrency control mechanism. Additionally, we have been inspired in our model from the **F**eedback **C**ontrol **S**cheduling **A**rchitecture (FCSA) [4] in order to support unpredictable workload. The proposed model imports stereotypes from MARTE [5] (**M**odeling and **A**nalysis of **R**eal-**T**ime and **E**mbedded systems) and UML-RTDB [6] (A **UML** profile for modeling **R**eal-**T**ime **D**ata**B**ases) profiles to represent real-time aspects on one hand, and to express RTDB features on the other hand.

The remainder of this paper is organized as follows. Section 2 presents related work in real-time object-oriented modeling. Section 3 provides a brief background information on MARTE, UML-RTDB and FCSA. Section 4 describes our RTO-RTDB object model. Section 5 describes the ongoing implementation of the RTO-RTDB model. Section 6 concludes by summarizing the contribution of this work, and discussing some future work.

2 Related work

RTDB research often uses the object-oriented paradigm [7]. However, no agreed-upon real-time object-oriented data model is available at this time.

DiPippo and Ma [8] describe the RTSORAC model which incorporates features that support the requirements of a RTDB into an extended object-oriented model. It extends a traditional object model with objects that contains real-time attributes, methods, constraints and compatibility functions. Each attribute includes its age, temporal consistency constraints, and imprecision. Methods are composed of a set of arguments, sequence of operations, execution times and operational constraints. Operations in methods are further described by deadlines and worst case execution times. The compatibility function provides a mechanism to express method concurrency in objects. DiPippo and Ma [8] do not describe the schedulability or the timeliness aspect of their model. They state that the constraints, real-time attributes and the compatibility function impose a higher degree of concurrency control in objects that allows the system to potentially meet its timing constraints. Fur-

thermore, imprecise computation encapsulated within the RTSORAC object is defined in the context of Epsilon Serializability (on transactions) [9], and does not supports the notion of quality of data introduced in [10] which allows a robust and controlled behavior of real-time databases during the transient overloads, based on feedback control real-time scheduling [4].

Stankovic and Son [11] describe the BeeHive model where the user sees the database as a set of BeeHive objects, a set of transactions, and a set of rules. Objects are used for modeling entities in the real world, transactions are used to specify application requirements and to execute the functionality of the application, and rules are used for defining constraints and actions to be taken. The BeeHive object model has some similarity in terms of the structure of objects to the RTSORAC object model [8]. In BeeHive, transaction execution times should be determined offline for admission control. This approach is only applicable to a limited set of real-time data services in which transactions and their arrival data access patterns are known in advances [12].

In [13], Perkusich et al. describe the G-CPN model which is based on object-oriented concepts and colored Petri nets. The G-CPN notations incorporates the notions of module and system structures in Petri nets. Each modules is an object with its own state and behavior, that provides a set of services or operations named methods that can be invoked by other modules of the system. Petri nets have shown to be useful to describe systems, allowing formal verification, and the execution of the model to obtain the behavior of a modeled system. However, to describe complex systems using Petri nets is not feasible, because the description trends to be very large and too difficult to comprehend and analyze.

The RODAIN model described by Taina and Raatikainen [14] is a real-time object-oriented database architecture for intelligent networks. The principal item in RODAIN real-time object model is an object. It can be either a regular object or a real-time object. Real-time objects consist of real-time attributes and real-time operations. A real-time attribute defines attribute access time that can be used to support different serializability levels. A real-time operations defines an estimate of the operation execution time that can be used for transactions scheduling. Real-time object are referenced by real-time transactions which are also real-time objects. They can be either persistent or transient. In RODAIN, the ultimate objective is to design and to specify a real-time object-oriented database architecture for telecommunications applications. Moreover, the authors have developed a data model tailored for the needs of telecommunications. Although RODAIN allows for the specification of real-time, object-oriented, and fault-tolerant database model, it is not general enough to be used to describe real-time objects in any real-time applications.

In [6], Idoudi et al. describe a real-time object-oriented data model. Each real-time object has four components: a set of real-time attributes, a set of real-time methods, a mailbox, and a local controller. Idoudi et al. [6] do not describe the schedulability or the concurrency aspects of their model. They said that the local controller component manages concurrency and schedulability aspects and do not specify how this is done.

In [15], Kim describes TMO model where objects are considered as time-triggered and message triggered objects. TMO consists of object data stores and service methods. The former are composed of lockable segments each containing data members. The latter include spontaneous methods that are only called from hardware interrupts. A message passing mechanism carry out all objects interaction. A server object guarantees timeliness by relying on the underlying hardware and the operating system to execute the service methods within its timing constraints. TMO qualifies the basic parameters for modeling a real-time object model. But the success of this model depends on the designers' knowledge of the underlying hardware and the operating system.

The object-oriented data models presented in this paragraph offer solutions to manage concurrency, schedulability, fault-tolerance of RTDBs. However, they do not specify concepts to express on the one hand quantitative features such as deadline and period and, on the other hand qualitative features that are related to communication and behavior. Moreover, they do not specify temporal and non-functional properties. Additionally, the presented models have mostly been described using natural language or logic based formalism which are not easily understood by an inexperienced designer. To remediate to this difficulty, the solution is using an expressive visual notation based on UML [16] to specify a real-time object-oriented database model. This improves the model specification because UML allows to easily visualize, define and document the artefacts of the system under development.

3 Background

This section provides background information on MARTE, UML-RTDB and FCSA which are used to describe our RTO-RTDB object model.

3.1 The MARTE Profile

MARTE is an industry standard of the OMG (**O**bject **M**anagement **G**roup) for model-driven development of embedded systems [5]. It defines extensions that provide high-level modeling concepts to deal with real-time and embedded features modeling as well as specific modeling artifacts to be able to describe both software and hardware execution supports. MARTE profile is organized around three main packages. The first defines the foundational concepts used in the real-time and embedded domain. It provides basic model constructs for non-functional properties, time and time-related concepts, allocation mechanisms and generic resources, including concurrent resources. These founda-

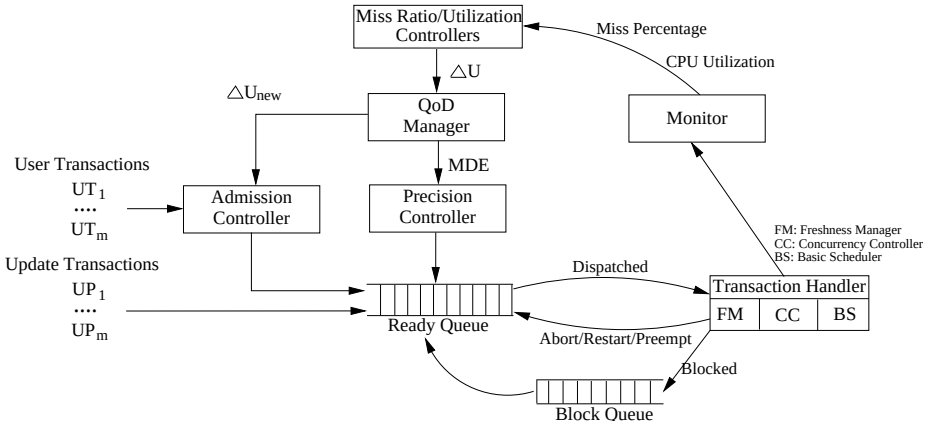

Figure 1. A Feedback Control Scheduling Architecture (FCSA) [10]

tional concepts are then refined in two other packages to respectively support modeling and analysis concerns of real-time embedded systems. The second package addresses model-based design. It provides high-level model constructs to depict real-time embedded features of applications, but also for enabling the description of detailed software and hardware execution platforms. The third package addresses model-based analysis. It provides a generic basis for quantitative analysis sub-domains. This basis has been refined for both schedulability and performance analysis.

3.2 The UML-RTDB Profile

UML-RTDB is an UML profile that allows the design of structural model for a RTDB [6]. It supplies concepts for RTDB modeling such as real-time attributes, real-time methods, and real-time classes. UML-RTDB stereotypes extend metamodel classes with specific sensor and derived attributes, specific periodic, sporadic, and aperiodic methods, and a specific real-time class.

3.3 Feedback Control Scheduling Architecture

The general outline of the FCSA is given in Figure 1. A FCSA consists of several components. In our study, the component we are interested to are the admission controller and the transaction handler. The admission controller is used to avoid the system overload by rejecting some user transactions when needed. Transaction handler provides a platform for managing transactions. It consists of a concurrency controller, a freshness manager, and a basic scheduler. Transactions are scheduled by a basic scheduler in the ready queue. The freshness manager checks the data freshness before a transaction accesses it. The concurrency controller ensures the concurrent execution of transactions (i.e. serializability).

4 The RTO-RTDB Object Model

A RTDB models an external environment that changes continuously. It is designed to be kept in shared main memory for fast and predictable access [1]. Real-time objects are the RTDB entities. They represent dynamic entities of time-critical dynamic systems in the real world. Our RTO-RTDB is an extension of the real-time object as used in [6], on which we incorporate time-constrained data, time-constrained methods and concurrency control mechanisms. As illustrated in Figure 2, each RTO-RTDB is made of five main components: (i) a set of classical attributes, (ii) a set of real-time attributes, (iii) a set of real-time methods, (iv) a set of real-time behaviors, and (v) a local controller.

One of the most important issues in the design of RTDBs is the concurrency control component. Its objectives are (i) to control the interactions between concurrently transactions and (ii) to maintain the database consistency. To handle this essential property of RTDBs, we associate to each RTO-RTDB a local concurrency control mechanism, named *local controller*, that manages the concurrent execution of its methods. A *local controller* is made of four components: (i) a state controller, (ii) a deadline controller, (iii) a freshness controller, and (iv) a concurrency controller (cf. Figure 2).

Because of the dynamic nature of the real world, more than one transaction may send requests to the same RTO-RTDB object. Concurrent execution of these transactions allows several methods to run concurrently within the same object. Since the workload of an RTO-RTDB object cannot be precisely predicted, it can become overloaded and thereby cause temporal violations of timing constraints. To support this issue, we based our RTO-RTDB object model on feedback control real-time scheduling theory. Using feedback control has shown to be very effective for a large class of real-time systems that exhibit unpredictable workload [10]. To the best of our knowledge, this is the first paper on *real-time object model* definition using feedback control architecture.

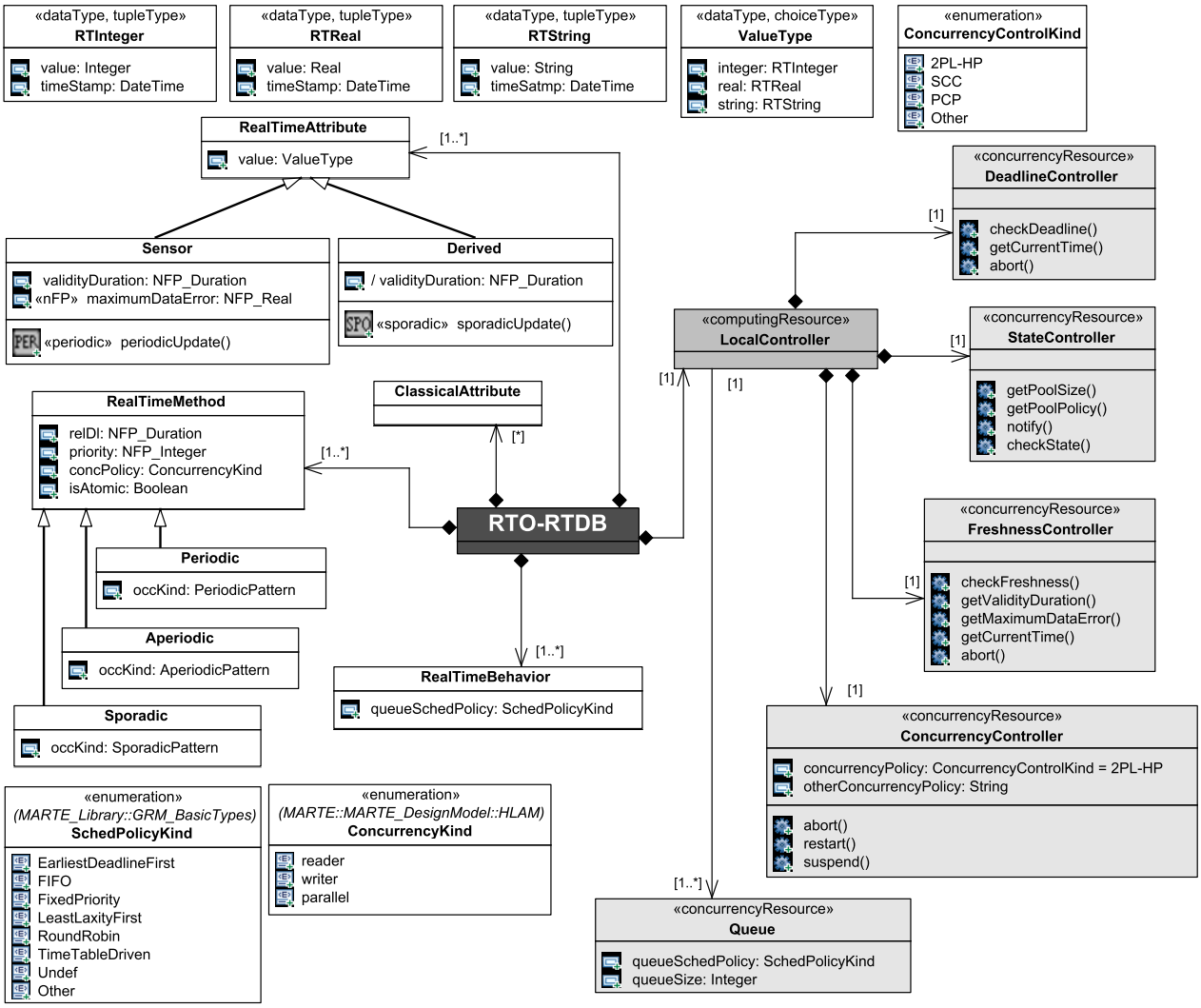

Figure 2. The RTO-RTDB object model

A RTDB is a collection of RTO-RTDB objects which are used to model time-critical dynamic systems in the real world. Each object has some internal state which is protected by the object abstraction. The only way that objects can be accessed by transactions is to invoke the methods defined by objects. Objects are encapsulated in the class abstraction which facilitates modular management of resources. We focus, in this paper, on modeling the RTO-RTDB object according to two views:

- A static view, which describes the entities, their relationships, the manipulated data that must be stored in the RTDB, and the different types of methods.
- A dynamic view, which describes the invocations of methods between the identified entities. We focus in this view on modeling the interactions between the *local controller* components.

We illustrate our proposal on a **F**reeway **T**raffic **M**anagement **S**ystem (FTMS), which consist of a large collection of data describing the current traffic state. This include road segment information such as traffic density (i.e. number of vehicles in a road segment), occupancy information, speeds and lengths of vehicles. The current traffic state is obtained from the essential sources: inductance loop detectors and supervision cameras. In fact, vehicle detector stations use inductance loops to measure speeds and lengths of vehicles, traffic density and occupancy information. This processed data is then transmitted at regular time intervals to the central computer system. Whereas, the supervision cameras are used to confirm the data received through the vehicle detector stations and to provide information on local conditions which affect the traffic flow. The computer system uses the acquired data stored in a real time database to monitor traffic and identify traffic incidents, when they occur. In our work, *Road Segment* and *Vehicle* are modeled as RTO-RTDB objects. Figure 3 shows the class diagram our FTMS application.

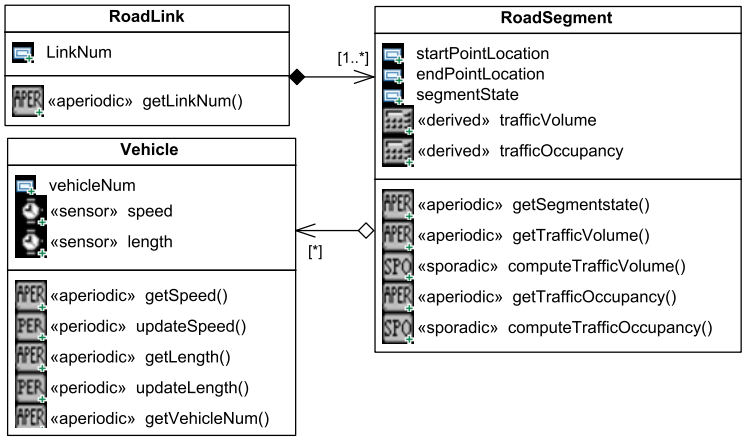

Figure 3. FTMS Class Diagram

4.1 Static View of The RTO-RTDB Object Model

To model our RTO-RTDB object model (cf. Figure 2), we use the class diagram of UML associated with sub-profiles of MARTE [5] (i.e. GRM, NFP, and VSL) and the UML-RTDB profile proposed in [6].

From GRM (Generic Resource Modeling) sub-profile, we import two stereotypes: *computingResource* and *concurrencyResource*. A *computingResourc*e represents either virtual or physical processing devices capable of storing and executing program code. A *concurrencyResource* is a resource that is capable of performing its associated flow of execution concurrently with others.

From NFP (Non-Functional Properties) modeling sub-profile, we import the *nfp* stereotype which extends the *Property* metaclass and shows the attributes that are used to satisfy non-functional requirements.

From VSL (Value Specification Language) package of MARTE, we import two stereotypes: *tupleType* and *choiceType*. An *tupleType* combines different types into a single composite type. The parts of a *tupleType* are described by its attributes, each attribute have a name and a type. A *choiceType* generates a data type each of whose values is a single value from any of a set of alternative data types.

From MARTE library, we import:

- The *DateTime* primitive type: it defines an instant of time in calender format.
- The *NFP_Duration* datatype: it allows the definition of time unit.
- The *NFP_Real* datatype: it specifies a real value of a non-functional property.
- The *PeriodicPattern* (respectively *AperiodicPattern* and *SporadicPattern*) datatype: it is a *tupleType* that contains the parameters that are necessary to specify a periodic (respectively aperiodic and sporadic) pattern.
- The *SchedPolicyKind* enumeration: it defines the kinds of scheduling policies (EarliestDeadlineFirst, LIFO, FixedPriority, etc.).

From UML-RTDB we import two stereotypes: *periodic* and *sporadic* which extend the *Operation* metaclass. Periodic (respectively sporadic) declares periodic transaction (respectively sporadic transaction) in a class diagram.

From UML-RTDB model library, we import three data types: RTInteger, RTReal, and RTString which describe the type of real-time attributes value and an enumeration type, called *ConcurrencyControlKind*, that defines concurrency control policies (i.e. 2PL-HP (Two Phase Locking-High Priority, PCP (Priority Ceiling Protocol), SCC (Speculative Concurrency Control), etc.). The following subsections detail our RTO-RTDB object model.

4.1.1 Real-time attributes

Data objects are classified into either real-time or non real-time data. A non real-time data is a classical data found in conventional databases, whereas a real-time data has a validity duration beyond which it becomes useless. We characterize the RTO-RTDB object model by two types of attributes: classical attributes and real-time attributes (cf. Figure 2). Classical attributes are used to store a non real-time data. As shown in Figure 3, we characterize the *Road Segment* object by three classical attributes, namely *startPointLocation* , *endPointLocation* and *segmentState*. The *RealTimeAttribute* class represents a real-time data. It incorporates fields to support logical constraints, temporal constraints, and quality of service constraints. The *RealTimeAttribute* class has an attribute: value, containing on one hand the final attribute value captured by the last update correspondent method, and the time at which the attribute's value was last updated on the other hand (i.e. timestamp). Real-time data are classified into sensor or derived data. Thus, these two types of data constitute the variations of

the *RealTimeAttribute* abstract class. In fact, a sensor attribute is used to store a sensor data which must be periodically updated in order to closely reflect the real world state of the application environment. It is characterized by two properties: validity duration and maximum data error. Validity duration indicates the amount of time during which the attribute value is considered valid. We use the MARTE *NFP_Duration* datatype as a type for the validity duration feature. Maximum data error indicates the maximum deviation tolerated between the current attribute value and the updated value. It is of type *NFP_Real*. Maximum data error represents a non-functional property specifying the upper bound of the error. We propose to associate the *nfp* stereotype of MARTE profile to the maximum data error attribute. The sensor class has a periodic update operation (i.e. *periodicUpdate()*) which is stereotyped by the UML-RTDB *periodic* stereotype and refers to a periodic operation of the class that owns the sensor attribute. The role of this periodic operation is to update the *value* field of the *sensor* attribute. For example, we characterize the *Vehicle* object by two sensor attributes, namely: *speed* and *length*. These attributes are periodically updated to reflect the state of a *Vehicle* instance. A derived attribute stores a derived data. The refreshment of each derived data is required every time one of the sensor data is updated. A derived attribute has a validity duration property, which is calculated as the intersection of validity duration of every used sensor data, and a sporadic update operation (i.e. *sporadicUpdate()*) which is stereotyped by the UML-RTDB *sporadic* stereotype and refers to a sporadic operation of the class that owns the derived attribute. This sporadic operation is used to update the *value* field of the *derived* attribute. We characterize the *Road Segment* object by two derived attributes: *trafficVolume* and *trafficOccupancy*.

4.1.2 Real-time methods

We consider a method execution as a transaction which is composed of one or many sub-transactions (a method can call other methods)[17]. We classify the RTO-RTDB object model methods into three classes: periodic methods, sporadic methods, and aperiodic methods (cf. Figure 2). Since the periodic, sporadic and aperiodic methods have the same structural characteristics (deadline, priority, concurrency policy, etc.), we propose an abstract class, called *RealTimeMethod*, in order to factorize these characteristics. The *relDl* attribute specifies the deadline of a method execution. The *priority* attribute specifies the priority order of a transaction. The *RealTimeMethod* stereotype factorizes also two properties: *isAtomic* and *concPolicy*. When the value of *isAtomic* is *true*, the method execution is executed as one individual unit. This fact coincides with the *atomicity* property of a transaction. However, in RTDB, this feature is relaxed in order to allow the validation of a transaction even if only a part of its actions have been executed [18]. So, in our work, we consider that the value of the *isAtomic* property is always *false*. The *concPolicy* property specifies the concurrency policy of a transaction. The values of this property may be: *reader*, *writer* or *parallel*. A *reader* transaction implies that multiple calls from concurrent transactions may occur simultaneously and will be executed simultaneously if there is no writer transaction using one or more data that the *reader* transaction needs. A *writer* transaction implies that multiple calls from concurrent transactions may occur simultaneously and will be treated as soon as concurrency on data allows its execution. A *parallel* transaction is a transaction whose actions do not use any data of the database in reading mode nor in writing mode. In our work, we consider that for an update transaction, which can periodic or sporadic, the *concPolicy* property is *writer*. The *occKind* attribute indicates the arrival transaction specification (i.e. periodic, aperiodic, or sporadic). For a periodic (respectively aperiodic and sporadic) transaction, the value of *occKind* property is equal to *PeriodicPattern* (respectively *AperiodicPattern* and *SporadicPattern*). We characterize the *Vehicle* object by a set of periodic methods such as *updateSpeed()* which periodically carries out write operations of *value* and *timeStamp* fields of the *speed* attribute. We characterize the *Road Segment* object by two sporadic methods: *computeTrafficVolume()* and *computeTrafficOccupancy()* which sporadically update the *value* and *timeStamp* fields of *trafficVolume* and *trafficOccupancy* attributes.

4.1.3 Real-time behaviors

The only way that objects can be accessed by transactions is to invoke the methods defined by objects. Thus, the invocation of methods constitute all messages received by the object. In this context, all processing are triggered by message arrivals. For each received message, a behavior is defined. Thereby, each object has one or several real-time behaviors which defines a message queue for saving messages received by the object and waiting to be processed. Additionally, each real-time behavior may specify the scheduling policy kind of its queue (i.e. EarliestDeadlineFirst, LIFO, FixedPriority, etc.) through the *queueSchedPolicy* property.

4.1.4 Local controller

The *local controller* manages concurrents execution of methods. When receiving a new message, if concurrency constraints allow this message to be processed, the *local controller* checks the timing constraint of the message (deadline, period, etc.) and proceeds to schedule it. Thus, it attaches a thread to this message and starts it. The *local controller* is in charge of: managing the messages received by the object, creating and destroying threads attached to each message processing, and managing the concurrency constraints. Beyond the concurrency aspects, computing aspects are also included in the *local controller*. For instance, checking deadlines or allocating threads are aspects of computing. Hence, the *local controller* is stereotyped by

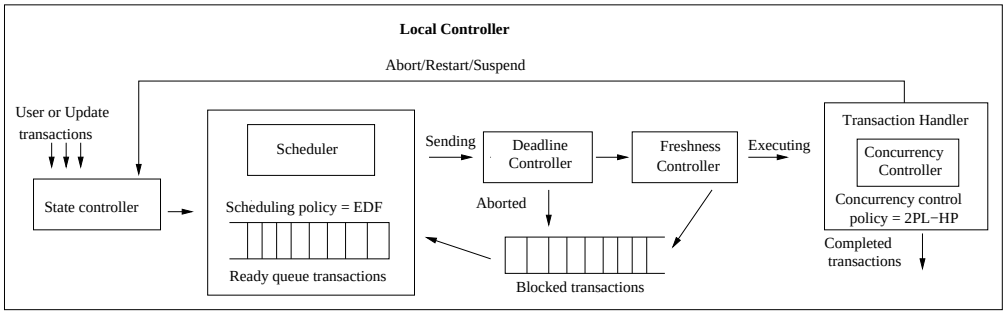

Figure 4. RTO-RTDB-FCSA

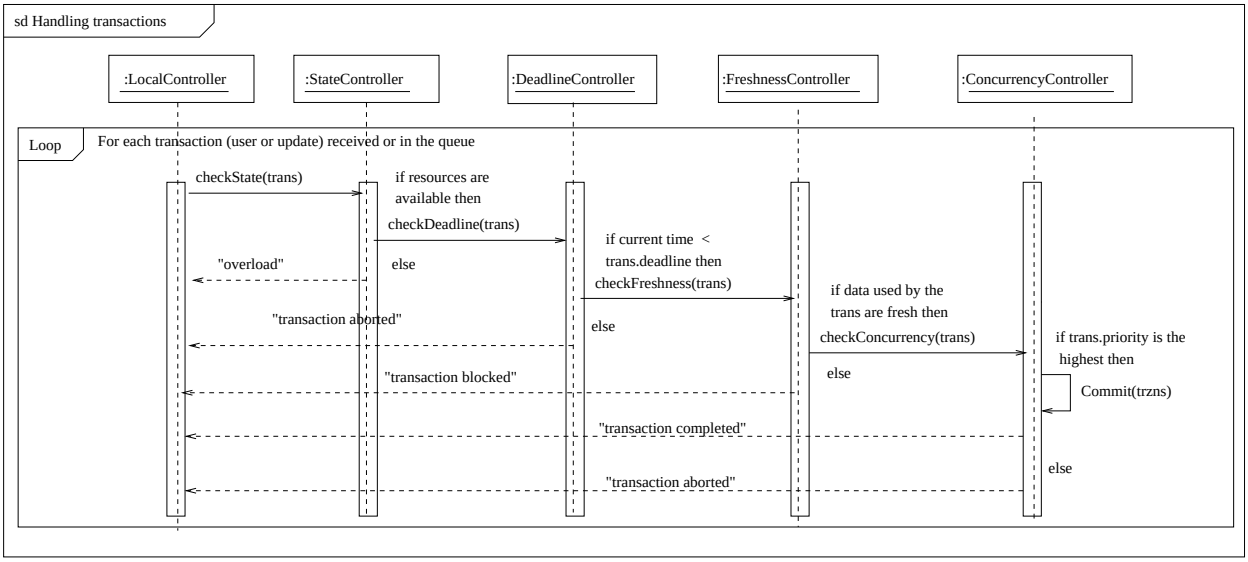

Figure 5. Dynamic view of the RTO-RTDB

computingResource imported from the GRM sub-profile of MARTE in order to indicate that it is capable of executing and performing computation. Additionally, the *local controller* is made of four components: state controller, deadline controller, freshness controller, and concurrency controller. Each component can perform its tasks concurrently with the others components. So, we assume that every components is stereotyped *concurrencyResource* imported from MARTE GRM sub-profile. In our work, the scheduling algorithm adopted by the *local controller* is Earliest Deadline First (EDF). For that purpose, we import from MARTE library the *SchedPolicyKind* enumeration which defines the most common kind of scheduling policy. Add to that, the concurrency control protocol adopted by the *local controller* is 2PL-HP. In 2PL-HP, all data conflicts are immediately resolved by aborting lower priority transactions. Thereby, we import from UML-RTDB model library the *ConcurrencyControlKind* enumeration type that defines concurrency control policies.

As mentioned in the introduction of this paper, the RTO-RTDB object model is based on the FCSA. Figure 4 depicts the RTO-RTDB-FCSA approach where the solid arrows represent the transactions flows. The majority of RTO-RTDB-FCSA components exist in the classical FCSA [10], but they are adapted as follows:

- ***Scheduler***:Transactions are scheduled according to their priority. The priority of a transaction depends on its deadline. Hence, we apply the EDF policy with respect to transaction priority. A lower priority transaction can be scheduled if there are no ready transactions with higher priority to schedule.

- ***State controller***: It is used to avoid RTO-RTDB overload. In fact, the RTO-RTDB has a pool of schedulable resources. When no schedulable resource is available, the RTO-RTDB may be overloaded. When receiving a message, the state controller checks if it can be handled. Otherwise, the message is rejected.

- ***Deadline controller***: It controls transactions validity [19, 20] . If the current time is greater than transaction deadline, transaction will be aborted. Otherwise, if

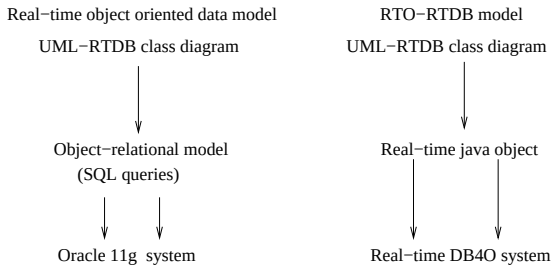

Figure 6. Mapping of the models to database systems

the verification step succeed, then the transaction is transferred to the freshness controller.

- ***Freshness controller***: It checks the freshness of acceded data just before a transaction commits. This way, the data accessed by committed transactions are always fresh at commit time. If the accessed data is fresh, transaction can be executed. Otherwise, the transaction will be blocked.

- ***Concurrency controller***: The main objective of this component is to verify the concurrency constraint between transactions. If it detects a conflict, it aborts the transaction having the lowest priority.

4.2 Dynamic View of The RTO-RTDB Object Model

An RTO-RTDB own one or several behaviors. A message queue for storing incoming messages is defined for each of these behaviors. As illustrated in Figure 5, the RTO-RTDB object receives messages in its queues awaking its *local controller*. The former checks the availability of resources through the *state controller* component. This latter is used to avoid RTO-RTDB object overload by reject some user (or update) transactions. When schedulable resources are available, the *deadline controller* checks the timing constraint attached to the messages and select one message following a scheduling algorithm depending on the type of the time constraint (i.e. deadline in our work), for instance the EDF scheduling policy. The *freshness controller* checks the freshness before a transaction accessed a data item. It blocks a user transaction if the target data item is stale. Based on 2PL-HP protocol, the *concurrency controller* ensures the concurrent transactions serializability. In case of conflict between transactions, when a higher priority transaction uses the data item, transactions with lower priority will be blocked.

5 Implementation

As mentioned in the previous section, RTO-RTDB is based upon an earlier model [6] called *real-time object-oriented data model*. Figure 6 illustrates the prototype implementation of the *real-time object-oriented data model*, and shows the parallel implementation of RTO-RTDB. The implementation of the *real-time object-oriented data model* is provided by an UML-RTDB class diagram, which allows the specification of objects and relationships [6]. These specifications are mapped to an object-relational model by means of SQL queries. These latter are executed in the Oracle 11g database management system.

A similar approach is proposed for implementation of RTO-RTDB. There are two main efforts involved with this implementation. First is the extension of DB4O [21], which is an open source object database, to support objects and transactions that have real-time characteristics. Thus, the real-time version of DB4O will replace Oracle 11g as the underlying database system. Second is the mapping of the UML-RTDB class diagram to real-time java objects to support characteristics of the RTO-RTDB model that do not appear in the *real-time object-oriented data model*. These characteristics include the *local controller* component for concurrency control, extended data type to support time, and the incorporation of time into the specification of transactions. Finally, we must automate the mapping of each components of an object or relationship to the attributes and methods of real-time java objects with library calls to real-time version of DB4O.

The effort to extend DB4O for real-time will involve modification or replacement of many of the components of DB4O. As it is an open-source project, we have had the opportunity to gain a full understanding of its modular implementation structure. We have recently begun to implement a prototype *Object Manager* capable of supporting an RTO-RTDB object model.

6 Conclusion

This paper has provided a general model for real-time object-oriented databases. Existing real-time object-oriented database model propositions still require high skill in RTDB features and offer restricted sources of concurrency. In order to overcome these limitations, the RTO-RTDB concept has been introduced. RTO-RTDBs ensure a high-level modeling of RTDBs providing parallelism at the system level because RTO-RTDBs are concurrent, and at the object level because RTO-RTDB operations may be executed concurrently depending on how they impact the object structural features. Moreover, unlike other propositions, the operational semantics of RTO-RTDB has been clearly described.

We are currently working on the complete implementation of our RTO-RTDB object model in order to add very positive evidences of its potential benefits.

References

[1] K. Ramamritham, Real-Time Databases, *Distributed and Parallel Databases*, 1(2), 1993, 199-226.

[2] K. Ramamritham, S. H. Son and L. C. DiPippo, Real-Time Databases and Data Services, *Real-Time Systems*, 28(2-3), 2004, 179-215.

[3] A. Bestavros, K. J. Lin and S. Son, Advances in Real-Time DataBase Systems Research, *Real-Time Database System: Issues and Applications*, (Kluwer Academic Publishers, 1997) 1-14.

[4] C. Lu, J. A. Stankovich, G. Tao and S. H. Son, Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms, *Real-Time Systems*, 23(1-2), 2002, 85-126.

[5] OMG, A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, version 1.0, formal/2009-11-02, 2009.

[6] N. Idoudi, N. Louati, C. Duvallet, B. Sadeg, R. Bouaziz and F. Gargouri, A framework to model real-time databases, *Int. J. of Computing and Information Sciences*, 7(1), 2010, 1-11.

[7] W. Kim, Object-oriented database systems: promises, reality, and future, in W. Kim (Ed.) *Modern database systems*, (New York: Addison-Wesley Publishing Co., 1995) 255-280.

[8] L. C. DiPippo and L. Ma, A UML package for specifying real-time objects, *Computer Standards and Interfaces*, 22(5), 2000, 307-321.

[9] K. Ramamritham and C. Pua, A Formal Characterization of Epsilon Serializability, *IEEE Transactions on Knowledge and Data Engineering*, 7(6), 1995, 997-1007.

[10] M. Amirijoo, J. Hansson and S. H. Son, Specification and Management of QoS in Real-Time Databases Supporting Imprecise Computations, *IEEE Transactions on Computers* , 55(3), 2006, 304-319.

[11] J. A. Stankovic and S. H. Son, Architecture and Object Model for Distributed Object-Oriented Real-Time Databases, *Proc. 1st Int. Symp. on Object-Oriented Real-Time Distributed Computing*, Kyoto, Japan, 1998, 414–424.

[12] S. Kim, S. H. Son and J. A. Stankovic, Performance Evaluation on a Real-Time Database, *Proc. 8th IEEE Real Time Technology and Applications Symp.*, San Jose, CA, USA, 2002, 253-265.

[13] M. L. B. Perkusich, M. Fatima, Q.V. Turnell and A. Perkusich, Object-Oriented Real-Time Database Design Based on Petri Nets, *Proc. of Int. Workshop on Active and Real-Time Database Systems*, 1995, 104-121.

[14] J. Taina and K. Raatikainen, RODAIN: a real-time object-oriented database system for telecommunications, *Proc. of Int Workshop on Databases: active and real-time*, New York, USA, 1997, 10-14.

[15] K. Kim, *Object Structures for Real-Time Systems and Simulators*,*J. of IEEE Computer*, 30(8), 1997, 62-70.

[16] OMG, Unified Modeling Language (UML), Infrastructure, v2.4, ptc/2010-11-16, 2011.

[17] J. Lee, S. H. Son and M. J. Lee, Issues in Developing Object-Oriented Database Systems for Real-Time Applications, *Proc. of the IEEE Workshop on Real-Time Applications*, Washington, DC, USA, 1994, 136-140.

[18] D. Agrawal, J. L. Bruno, A. E. Abbadi and V. Krishnaswamy, Relative Serializability: An Approach for Relaxing the Atomicity of Transactions, *Proc. of the 13th ACM Symp. on Principles of Database Systems*, 1994, 139-149.

[19] C. J. Date, *An Introduction to Database Systems*, (New York: Addison-Wesley Publishing Co., 1985).

[20] C. Liu, J. Leyland and W. James, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *J. of ACM*, 20(1), 1973, 46-61.

[21] J. Paterson, S. Edlich and H. Hörning and R. Hörning, *The definitive guide to DB4O*, (Apress, 2006).